Разработка простейшего файлового антивируса на python

Оглавление

1.1. Вирусы

- Вирус – специализированное ПО, способное к несанкционированному внедрению, внесению изменений, создания своих функциональной идентичных копий
- Полиморфная и функциональная идентичность – понятие функциональной идентичности было введено из-за наличия класса полиморфных вирусов, экземпляры которых не имеют сходства, но выполняют идентичные действия
- Троянская программа – вредоносное ПО, которое не имеет возможностей к самовоспроизведению и нацелено, как правило, на несанкционированные деструктивные действия

1.1. Антивирус - специализированное антивредоносное ПО, разрабатываемое для защиты от киберугроз(вирусы, спам, фишинг, вредоносные URL). Служит для предотвращения, обнаружения и удаления вредоносного ПО.

1.2. Методы проверки

- Сигнатурный анализ – поиск характерных для вируса свойств при сравнение каждого отдельно взятого файла, другое название hash-scan. Обычно проверка происходит путём сравнения сигнатуры файла со словарём, заранее составленным из сигнатур известных вирусов. Файл считается заражённым если его сигнатура в ходе сканирования обнаружена в соответствующей базе данных. Метод уязвим перед новыми вирусами, а также перед полиморфными или ширяющимися вирусами
- Эвристический анализ – оценивает функционал файла или программы с помощью шаблонов для обнаружения подозрительной активности. Минусом данного подхода является возможность ложного срабатывания при сканировании. Для выявления загрузочного вируса, расположившегося в MBR, считывается загрузочная запись двумя

путями: WinAPI ReadFile и direct disk access — DDA driver. После чего сравниваются оба буфера, при их несовпадении с высокой степенью вероятности можно судить о наличии вируса, наиболее часто это применимо для руткитов

- Регламентация порядка работы с файлами, программами и службами
- Обнаружение в песочнице – использование виртуального пространства для сканирования поведенческого отпечатка во время выполнения программы

Следует отметить, что для корректной работы антивирусу необходимо не только иметь возможность рассчитывать хеш-сумму файла, но также и обращаться к расшифровщику, если файл закодирован или же работать с известными архивами. Упаковщики могут использоваться для попытки обхода сигнатурного анализа, сжимая исходный код таким образом, чтобы он был необходим во время распаковки файла. Таким образом сканируемый файл будет совершенно не похож на имеющий в блек-листе. Также вредоносный код может быть подвергнут шифрованию, что в свою очередь приводит к изменению его сигнатуры.

Другим вариантом может быть перехват функций проверки антивирусом до их запуска function hooking, что в свою очередь приведёт к некорректности проверки. Схожим методом может служить и подмена памяти при выполнении сканирования, когда неудовлетворительный результата всегда подменяется на «успешно».

Обфускация(obfuscation) – метод обхода защиты, при котором затрудняется возможность чтения кода.

1.1. Задание

Создать антивирусную программу hash-scan на языке python выполняющую проверку тестовых файлов в текущей директории с помощью сравнения их хеш-сумм и последующего удаления «заражённых».

- Реализация получения hash файлов с помощью алгоритма sha256
- Запись полученных хеш-сумм в txt файл “HashList.txt”
- Запуск тестовой программы “FC”, выполняющей изменение файлов в директории
- Вычисление хеш-суммы файлов после их изменения
- Сравнение полученных хеш-сумм со словарём “VirusHashList.txt”

- Определение “changed” (хеш-сумма отличается от первоначальной) и “infected” (хеш-сумма находится в блек-листе) файлов
- Запись полученных результатов в report.txt в формате:

Origin hash:

1.txt - …

…

10.txt - …

Changed:

1.txt - …

Infected:

1.txt - …

- Удаление infected файлов из директории

1.2. Разработка на Python

Подключите необходимые для работы библиотеки

- Hashlib – представляет собой интерфейс для легкого хеширования сообщений с поддержкой OpenSSL. Он содержит множество методов, которые будут обрабатывать хеширование любого необработанного сообщения в зашифрованном формате. Предоставляет доступ к следующим алгоритмам: SHA1, SHA224, SHA256, SHA384, SHA512, SHA-3, MD5, blake2b, blaske2s
- Хеширование файлов выполянется следующим образом: hashlib.**file_digest**(*fileobj*, *digest*, */*)
- Zlib – библиотека предоставляющая доступ к алгоритмам: adler 32, crc32
- **import** hashlib – подключение библиотеки необходимой для получения доступа к методам шифрования, hashlib.sha256() – использование алгоритма шифрования
- **impost sys, import os – подключение** библиотек необходимых для работы с файловой системой, вводом и выводом и др.

Для обзора доступных методов шифрования в системе выполняется следующая команда

- print(hashlib.algorithms_available) – вывод доступных алгоритмов доступных в системе {'whirlpool', 'sha3-224', 'sha3-256', …}

Необходимо определить рабочую директорию для программы и тестовых файлов

- Получение текущей директории - current_dir = os.path.dirname

Определяем хеш файлов в директории и записываем в файл

- f = open('example.txt','r') - открыть файл из рабочей директории в режиме чтения, r+ - чтение и запись
- print(*f) - выводим содержимое файла, после открытия файла его необходимо закрыть - f.close(). Аналогично возможно использование конструкций try/finally, with и др.
- Открытие файла для записи - f.write(‘string’)
- Расчёт хеша файла - with open(file_name) as f:

  data = f.read()

  sha256hash = hashlib.sha256(data).hexdigest()

Следует учитывать кодировку содержимого файлов.

Полученные хеши заносим в файл для последующего сравнения

Запускаем утилиту, которая вносит изменения в некоторые файлы

Выполняем проверку тестовых файлов, обнаруживая «заражённые», внося их имена в соответствующий файл

В случае, если размер файлов для хеширования слишком велик необходимо использовать алгоритм с применением разбития на блоки:

```
def compute_sha256(file_name):
    hash_sha256 = hashlib.sha256()
    with open(file_name, "mod") as f:
        for chunk in iter(lambda: f.read(size of blocks), b""):
            hash_sha256.update(chunk)
    return hash_sha256.hexdigest()
```